AMENDMENTS TO THE CLAIMS

1-4. (Canceled).

5. (Previously Presented) A scheduling method for a load microinstruction, comprising:

if a new load microinstruction is admitted, predicting whether collision occurs between load microinstruction and an older store microinstruction,

if a collision is detected, determining whether data for the older store microinstruction is available in a store unit,

if data for the older store is not available, storing the load microinstruction in a scheduler with a marker indicating that scheduling of the load microinstruction is to be deferred.

6. (Original) The scheduling method of claim 5, further comprising storing a scheduler entry identifier of the older store with the load microinstruction.

7. (Original) The scheduling method of claim 5, further comprising scheduling the load microinstruction for execution after the marker is cleared.

8. (Original) The scheduling method of claim 7, further comprising scheduling other instructions dependent upon the load microinstruction to execute after the load microinstruction executes.

9. (Original) The scheduling method of claim 5, further comprising deferring scheduling of other instructions dependent upon the load microinstruction when scheduling of the load microinstruction is deferred.

10. (Previously Presented) The scheduling method of claim 5, wherein the store microinstruction is part of a plurality of microinstructions representing a store instruction, wherein a first of the plurality to transfer data to the store unit and a second of the plurality is the older store microinstruction, which is to calculate an address of the store instruction.

11. (Original) The scheduling method of claim 10, further comprising clearing the marker of the load microinstruction after the first store microinstruction executes.

12. (Original) The scheduling method of claim 10, wherein the prediction determines a collision between the load microinstruction and the second store microinstruction.

13. (Original) An execution unit for a processing agent, comprising:

a scheduler operating according to the method of claim 5,

a register file, and

a plurality of execution modules,

wherein the scheduler, the register file and the execution modules each are coupled to a common communication bus.

14. (Previously Presented) A scheduling method, comprising:

predicting whether a new load microinstruction collides with a first previously received store microinstruction when the new load microinstruction is admitted to a scheduler,

if a collision is detected, determining whether data for the older store microinstruction is available in a store unit if data for the older store is not available, storing the load microinstruction in the scheduler with a dependency pointer to a second previously received store microinstruction.

15. (Previously Presented) The scheduling method of claim 14, further comprising scheduling the load instruction for execution after the dependency pointer is cleared.

16. (Original) The scheduling method of claim 15, further comprising scheduling other instructions dependent upon the load instruction to execute after the load instruction executes.

17. (Original) The scheduling method of claim 14, further comprising deferring scheduling of other instructions dependent upon the load instruction when scheduling of the load instruction is deferred.

18. (Previously Presented) The scheduling method of claim 14, wherein the store microinstruction is part of a plurality of microinstructions representing a store instruction, wherein a first of the plurality to transfer data to the store unit and a second of the plurality is the older store microinstruction, which is to calculate an address of the store instruction.

19. (Previously Presented) The scheduling method of claim 18, further comprising clearing the dependency pointer of the load microinstruction after the first store microinstruction executes.

20. (Original) The scheduling method of claim 18, wherein the prediction determines a collision between the load microinstruction and the second store microinstruction.

21. (Original) An execution unit for a processing agent, comprising:

a scheduler operating according to the method of claim 14,

a register file, and

a plurality of execution modules,

wherein the scheduler, the register file and the execution modules each are coupled to a common communication bus.

22. (Previously Presented) A dependency management method, comprising, upon execution of a STD uop:

comparing an identifier of the STD microinstruction to dependency pointers of other microinstructions stored by a scheduler, and

clearing any dependency pointers that match the identifier.

23. (Previously Presented) The dependency management method of claim 22, wherein the identifier represents a location in the scheduler where the STD microinstruction is stored.

24. (Previously Presented) The dependency management method of claim 22, wherein the identifier represents a location in a store unit where data responsive to the STD microinstruction is stored.

25. (Previously Presented) The method of claim 22, wherein the STD microinstruction causes a transfer of data to a store buffer when executed.

26. (Previously Presented) The method of claim 22, wherein the STD microinstruction is paired with a STA microinstruction that, when executed, causes calculation of a store address.

27. (Previously Presented) A dependency management method, comprising:

decoding a store instruction as a plurality of microinstructions, including an STA and a corresponding STD microinstruction,

decoding a load instruction as at least one load microinstruction,

when a dependency between the STA microinstruction and the load microinstruction occurs, deferring scheduling of the load microinstruction until after the corresponding STD

microinstruction executes.

28. (Previously Presented) The dependency management method of claim 27, further comprising detecting the dependency between the STA microinstruction and the load microinstruction by comparing an identifier of the STD microinstruction to dependency pointers of other microinstructions stored by a scheduler, and clearing any dependency pointers that match the identifier.